2.5 SENSORS

A sensor is a system used to obtain information about another platform without its cooperation or permission. Sensors gather this information about an object by collecting and processing energy emanating from the object. An active sensor transmits energy and collects the reflection of that energy from the object, while a passive sensor collects existing energy emanating from the object. SWEG can model both active and passive sensors. Common types of sensors in DoD scenarios include radio frequency (RF) sensors (radars), electro-optical (EO) sensors, IR sensors, and ultraviolet (UV) sensors. SWEG can model these and other types of EM and acoustic sensors. SWEG has only two categories of sensors, acoustic and EM, it contains no subdivisions of the EM spectrum into EO, IR, RF, UV, etc. A user can define a sensor with frequencies in any portion of the EM spectrum or any portion of the acoustic spectrum. Susceptibility to sensors is defined similarly (see Section 2.3). The only difference in the way SWEG handles acoustic and EM sensors is in the signal transmission speed used to convert frequency to wave length. Within the simulation, signal transmission is assumed to be instantaneous for all types of sensors.

Sensors can be used to search for unknown, suspected, or known targets; they may be used to acquire or lock on a target to prepare for firing; they may be used to track a target in order to fire a weapon at it; or they may be used to guide a weapon to a target. SWEG can model all of these sensor uses; a sensor may be defined to be search only, track only, or search and track. A sensor with tracking capabilities is assumed to have at least three modes: acquisition (prior to lockon), track (after lockon, before firing), and guidance (after firing). SWEG allows a user to define different center frequencies and different sensing rates for the four possible sensor modes: search, acquisition, track, and guidance.

Data gathered by a sensor may consist of just one item such as range or azimuth or closing speed, or it may include more complete information such as target type and three dimensional position. SWEG allows the user to select the data items to be collected by a sensor type, although currently the user should always include both altitude and planar position because SWEG does not correctly represent approximations of either item. Other user-selectable data items that SWEG sensors can provide include target type, heading (including pitch), speed, age of platform, and information about the platform structure and systems. Data automatically obtained during a detection in SWEG are the time of the detection, the detected frequency (important for a passive sensor), identification friend or foe (IFF) results if the sensor has IFF capability, and tracking results if the detection occurs during track. Data obtained by sensors in the real world may include errors; however, in SWEG, any data collected by a sensor is accurate and is (initially) accurately perceived by the player that owns the sensor.

In order to obtain information about a target, a sensor must be able to detect it. Detection of a target occurs if the energy received from the target at the sensor receiver (signal) is great enough to be noticeable when compared to the other energy received there (interference). Detection depends on characteristics of the sensor, the target, the environment, energy transmission, and the geometric relationship of the sensor and the target. In SWEG, most of the sensor, target, and environment characteristics are user-defined. Relevant characteristics of the target and the environment are discussed in Sections 2.3 and 2.12 - 2.18, respectively. User-defined sensor characteristics used to determine detection include threshold, power, frequencies, directional gain (antenna pattern), receiver losses, receiver noise (or one-square-meter detection range), pulse

repetition frequency, and attenuation due to moving target indicator processing. Signal polarization and receiver bandwidth can also affect detection; however, in SWEG, these are used only to determine jamming effects.

In SWEG, as in many Department of Defense (DoD) models, detection is determined by comparing the signal-to-interference ratio (S/I) to a detection threshold. SWEG allows the user to specify the threshold values. A sensor threshold generally has some randomness associated with it; i.e., detection does not always occur whenever S/I is greater than the threshold, but will occur then with some probability. SWEG allows the user to define probabilities of detection to be used to determine detection after S/I exceeds the threshold. Sensors may require more than one hit (detection) in order to establish detection of a target; SWEG allows the user to specify a number of hits to establish target detection for any sensor. The probability of detection is tested only after the required number of hits occur. If this test is passed, the information is passed to the thinker associated with the sensor receiver.

For moving sensors, especially acoustic sensors, thresholds may change with movement speed; SWEG allows the user to define threshold changes that depend on speed and signal frequency for any sensor. For tracking sensors, SWEG allows a user to specify distinct thresholds and associated probabilities of detection for pre- and post-lockon, and to specify the time delay after lockon that the new threshold should be used. Tracking sensors may lose lockon while tracking and go into a coasting mode; if lockon is not regained, any enroute ordnance may abort. SWEG allows the user to define a probability of continuing track, and will put the sensor into coasting mode if a continue-track test is failed. Users can also define a maximum time that a tracker can remain in coast mode without causing enroute ordnance to abort.

Some sensors, primarily those that are used only for search, have antennas which rotate horizontally; other sensors, especially tracking sensors, may be pointed at a target. Any sensor in SWEG may be defined to be rotating (i.e., sweeping around in a plane parallel to the ground) or pointing as directly as possible at its target. Sensors may have various physical and operational limits, such as antenna slew limits, receiver frequency limits, or a maximum number of parallel tracks that a tracking sensor can handle at any one time. SWEG allows the user to define the last two listed. The user may also specify maximum and minimum elevation slew angles for the sensor; but not maximum or minimum azimuth slew angles, nor any limits on slew rates.

The user may define any number of any type of sensors on any platform. For active sensors, transmitters and receivers may be in separate locations and the user may specify the vertical offset of each from the platform location (in this case, SWEG uses the target signature associated with the average aspect angle of the two locations). SWEG also allows the user to specify pointing direction, operating frequencies, and operational status (on, off, or non-operational) of each individual sensor of any type on each individual platform.

2.5.1 Functional Element Design Requirements

This section contains the design requirements necessary to implement the simulation of sensors in SWEG.

a. SWEG will provide the capability for a user to define types of sensors that operate at any frequencies in the EM or acoustic spectrum, but no single sensor type in SWEG will operate in both spectra. SWEG will model instantaneous signal transmission; thus, the only difference in the way SWEG treats EM and acoustic sensors will be to use the appropriate signal transmission speed as necessary in the calculation of parameters.

b. SWEG will provide the capability for the user to define both active and passive sensor types. Two categories of passive sensor types will be modeled: those that detect active transmissions and those that detect passive emissions.

c. SWEG will simulate detection attempts only for those entities that have been explicitly identified by the user to be detectable by the sensor type. SWEG will simulate sensing chances at a user-defined rate whenever a detectable entity is within the maximum range of a sensor. The maximum range is defined by the user as part of a range-altitude capability profile described in Design Requirement i.

d. SWEG will determine the result of each sensing chance in four steps. First, line of sight and all user requirements for detection by the sensor (see Design Requirements c and i) are checked. Second, if these tests are passed, the ratio of the S/I received will be compared to a user-defined threshold. Third, SWEG will compare the number of times the threshold has been met to a user-defined required number. Fourth, if these tests are passed, SWEG will determine detection or no detection based on a user-defined probability. In the second step, SWEG will change the threshold based on sensor movement, if the user has so specified.

e. SWEG will use the standard energy signal transmission and reflection equation to calculate the S/I received. The following user-defined parameters will be included for active and passive sensors:

    1. gain of the receiver type
    2. internal receiver type losses
    3. receiver frequency
    4. receiver type noise

    In addition, SWEG will provide the capability for the user to calibrate the sensor by defining the detection range of that sensor type for a one-square-meter target. The equation may also include jamming as part of the interference (see Section 2.10)

    For active sensors, the following user-defined transmitter type parameters will also be used in the S/I calculation:

    1. transmitter type peak power
    2. transmitter type gain
    3. internal loss for transmitter type

In addition, for active sensors, the equation may include a Moving Target Indication (MTI) attenuation factor (see Design Requirement h).

f. SWEG will provide the capability for the user to define an antenna gain table for each type of sensor transmitter and each type of receiver. The table will provide antenna gain values for frequency, azimuth, and elevation intervals arbitrarily defined by the user (frequency is optional). These values will define a step function; i.e., no interpolation will be performed. The vertical offset of the antenna may also be defined by the user.

g. SWEG will provide the capability for the user to define the receiver frequency of a specific sensor in either of two ways; either by selecting a specific frequency between the minimum and maximum frequencies defined by the user for that receiver type, or by specifying that SWEG randomly select a frequency in the interval between them. If neither option is selected by the user, SWEG will use the center frequency for the receiver type. Frequencies may be different for each mode (search, acquisition, track, or guidance) of a sensor type. For active sensors, the frequency of the transmitter will be set to the frequency of the receiver.

h. For active sensor types, SWEG will provide the capability for the user to define MTI attenuation as a function of Doppler frequency intervals. In each sensing chance, the Doppler frequency will be calculated using the relative radial velocity of the sensor and the target and the following user-defined parameters:

1. pulse repetition frequency of the sensor type
2. frequency of the specific sensor

i. SWEG will disallow detections for those sensing chances that do not meet the following user specifications:

1. maximum and minimum radial speed limits
2. symmetric azimuth limits
3. maximum and minimum elevation limits
4. elevation slew limits
5. relative altitude limits for user-defined range intervals

j. The following data will automatically be obtained in a detection during a SWEG run:

1. time of detection
2. frequency of signal detected
3. IFF results if the user has specified IFF capability for this sensor type
4. tracking results if detection occurs during track

In addition, SWEG will accurately provide any of the following data for a detection if so instructed by the user:

1. target altitude
2. target x-y location
3. target azimuth
4. target heading (including pitch)
5. target speed
6. age of target from time of birth (start of scenario or disaggregation time)

7. number of *elements* on the target *platform*
8. *player* name
9. *platform* id

k. SWEG will provide the capability for the user to define a sensor as rotating or not rotating. Rotating sensors will be modeled as sweeping around in a plane parallel to the ground; at each sensing chance, targets always be modeled as in the main beam with respect to azimuth, but not necessarily with respect to elevation. Non-rotating sensor receivers are modeled as pointing as directly as possible at the target (subject to slew limits) for each sensor chance.

l. SWEG will provide the capability for a user to define sensor types that are search only, track only, or search and track. Search sensors will be used to provide information used in decision-making. Track sensors will be used in conjunction with weapon systems during an engagement. Users may specify whether or not each search and track sensor type can operate in both modes simultaneously (track-while-scan).

m. For each sensor type with tracking capabilities, SWEG will model three modes of operation; acquisition (prior to lockon), track (after lockon, before firing), and guidance (after firing). SWEG will provide the capability for the user to define separate thresholds and separate probabilities of detection (as described in Design Requirement d) for pre- and post-lockon modes of tracking sensors.

n. For each sensor type with tracking capabilities, SWEG will provide a capability to implement the following user-defined time restrictions:

1. delay from the decision to initiate an engagement until the first sensing chance is scheduled for the tracker.
2. length of time the pre-lockon threshold must be used before switching to the post-lockon threshold.
3. length of time a tracker can remain in coast mode (after losing lock) without causing enroute ordnance to abort.

o. For each sensor type with tracking capabilities, SWEG will limit the number of targets tracked simultaneously if the user has defined a maximum.

p. For each specific sensor receiver or transmitter in a scenario, SWEG will provide the capability for the user to define the following additional data:

1. status: on, off (may be changed to on by decision-making capabilities), or non-op (nonoperational for the whole exercise or until a user-defined time for a status change). If the user does not specify a status, SWEG will assume the sensor is on.
2. any number of scheduled status changes (game time and new status)
3. pointing direction at a specific platform or a specific geometric location or in a specific direction. If the user does not specify a pointing direction, SWEG will point the sensor in the same direction as the forward-facing direction of the platform.

q. SWEG will provide the capability for a user to define an emission code for each sensor transmitter mode that will identify it for use in correlating model actions with other simulations in a networked environment. (This requirement has

currently led to a violation of the precept that TDB and SDB instructions should contain no instructions specific to either a constructive or a virtual mode of operation).

2.5.2 Functional Element Design Approach

This section is not currently available.

2.5.3 Functional Element Software Design

This section contains a table and two software code trees which describe the software design necessary to implement the requirements and design approach outlined above. Table 2.5-1 lists most of the functions found in the code trees, and a description of each function is provided. Figure 2.5-1 depicts the path from *main* to *BSRVevent*, the top-level C++ function within the code for sensor events, and its major subfunctions. Figure 2.5-2 is the code tree for *BSRVevent* and its subordinate functions.

A functions subtree is provided within the figure only the first time that the function is called. Some functions are extensively called throughout SWEG, and the trees for these functions are in the appendix to this document rather than within each FE description. Within this FE, the functions in that category are *MITRcontrol* as well as all member functions in the C++ class *WhereIsIt*.

Not all functions shown in the figures are included in the table. The omitted entries are trivial lookup functions (single assignment statements), list-processing or memory allocation functions, or C++ class functions for construction, etc.

TABLE 2.5-1. Sensors Functions Table.

Function	Description
ailrip	schedules friendly players to notice death from attack
antgeom	calculates antenna pointing and relative angle data
badpro	processes a randomly failed thinker/processor systems
BaseHost:Run	runs all steps
begone	eliminates a perception of a platform
BSRVaimpt	determines aiming point for an antenna
BSRVcalculate	determines sensor result using signal/noise calculations
BSRVevent	controls sensor physical processing
BSRVfiresig	computes increase in signature due to weapon launch
BSRVgetsigtable	gets the appropriate signature table
BSRVinitialize	initializes physical sensor processing
BSRVmasking	performs masking calculations for sensor systems
BSRVnextchance	determines next sensing chance for a given target
BSRVnomore	processes end of sensing chances for this target
BSRVonechance	supervises sensor chance calculations
BSRVresults	prepares sensor chance results report

TABLE 2.5-1. Sensors Functions Table. (Contd.)

Function	Description
BSRVsignature	computes cross sectional area of a target
BSRVsnrerrors	senses target using seeker errors and target elevation
BSRVsystsensing	performs physical system sensing calculations
BSRVtgtsignal	determines total target signal power
BSRVtrack	determines sensor chance tracking results and effects
BSRVwrapup	finishes up this sensor chance
crslwc	determines line/circle crossing
crslwl	determines line/line crossing
crslwp	determines line/plane crossing
delswt	deletes engaged weapon buffer block from various lists
diefri	deletes friendly perception structures
ergatn	calculates attenuation for energy transmission
ergazel	retrieves table entry for gain from azimuth and elevation
erggar	calculates gain and range for energy transmission
featlos	determines if shapes interfere with line of sight
haltit	deletes movement data structures associated with platform
immol8	performs self-destruction of platform and possibly parent player
injcom	deletes destroyed comm system
injsnr	deletes destroyed sensor system
injure	deletes systems belonging to an element
jamcal	calculates jammer interference power at victim receiver
jetind	recycles indirect list entries and possibly top list
KILLperception	deletes a sensor-derived perception
limevent	eliminates unneeded event nodes data structure
limpendq	eliminates unneeded pending queue data structures
loschk	checks link of site between two objects
main	controls overall execution
MainInit	initiates processing
MainParse	controls parsing of user instructions
MITRcontrol	controls emitter system status changes
mutil8	deletes data structures associated with a platform
nayist	calculates the next sensing time
numerical	sorts address codes
program	main program for execution phase
redwood	adds new entry to or removes top entry from leftist tree
region	determines if a point is within a two dimensional region
RNDMgrn	generates a gaussian randomly distributed number
sasin8	performs bookkeeping involved with death of a player

TABLE 2.5-1. Sensors Functions Table. (Contd.)

Function	Description
semant	controls semantic processing of instructions
sigmov	determines effects of movement on sensor detection
simnxt	controls runtime execution
simphy	controls processing of physical events
simul8	controls semantic processing of runtime instructions
snropfreq	determines operating frequency for a specific chance
snropmode	determines operating mode for a specific chance
srhpro	searches table for interval containing a specific value
TAddrData::DelOccupant	deletes platform from the list of occupants at this address
TAddrData::GetJamInteractions	checks for jammer interactions with a communications device
TAddrData::GetParentData	retrieves the TAddrData object from the parent
TAddrData::GetShapeList	finds the shape list at a given address
TAddrData::TryDeletion	tries to delete a node in the address tree
TAddress::GetAddresses	retrieves a sorted collection of addresses between two points
TAddress::GetCellRadius	determines the radius of an address tree cell
TAddress::GetShapeList	creates a list of shapes that might affect line of sight
TAddress::InsertVertCodes	inserts address codes, including parents, for the given point
TMaster::GetPlatform	retrieves a platform from the direct access list
TMaster::GetPlayer	retrieves a player from the direct access list
TMaster::GetRandomTable	retrieves a random number table
TMBRpercplat	recycles storage associated with a perception of platform
TMemory::Allocate	allocates permanent storage
TMemory::AllocTemp	allocates temporary storage
TMemory::Deallocate	deallocates a list of blocks by using the address within the provided pointer
TMemory::DeallocFront	deallocates storage
TMemory::DoAllocate	allocates either permanent or temporary storage
TMemory::LLSTlength2	counts the number of list entries
TMemory::LLSTlistofflist	searches on list using a second list
TMemory::LLSTremove	returns a pointer to the block on the traversed list which matches the provided key
TMemory::LLSTsearch	searches a list
TMemory::LLSTsearchhard	searches a list using extra parameters
TPathEntry::GetBeforePoint	looks up the path point prior to given time
TTable::SearchInt	searches a table for a specific integer
TTerrain::EdgeMasklos	determines the masking of terrain edges
TTerrain::Elevation	returns the z-coordinate of point on surface at a specific location
TTerrain::FindTriangle	determines the terrain triangle for a point given an x, y coordinate pair

TABLE 2.5-1. Sensors Functions Table. (Contd.)

Function	Description
TTerrain::LineOfSight	determines if there is a line of sight between two objects
WhereIsIt::CalcFwdVector	calculates the local forward direction vector for platform
WhereIsIt::CalcHeading	determines heading for a platform given a time
WhereIsIt::CalcPitch	determines pitch for a platform given a time
WhereIsIt::CalcPosition	determines position for a platform given a time
WhereIsIt::CalcSpeed	determines speed for a platform given time
WhereIsIt::CalcUnitVel	determine unit velocity for a platform given a time
WhereIsIt::CalcUpVector	calculates the local up direction vector for platform
WhereIsIt::CalcVelocity	determine the local velocity vector at a specific time
yaeail	adds yet another entry to the scenario action item list

```
main
    |-BaseHost::Run
        |-MainInit
            |-program
                |-MainParse
                    |-semant
                        |-simul8
                            |-simnxt
                                |-simphy
                                    |-BSRVevent
                                        |-BSRVinitialize
                                        |-BSRVonechance
                                        |    |-BSRVsignature
                                        |    |    |-BSRVaimpt
                                        |    |    |-BSRVgetsigtable
                                        |    |    |-BSRVfiresig
                                        |    |-BSRVmasking
                                        |    |-BSRVcalculate
                                        |    |    |-BSRVtgtsignal
                                        |    |    |-BSRVsystsensing
                                        |    |-BSRVresults
                                        |    |    |-BSRVsnrerrors
                                        |    |-BSRVtrack
                                        |-BSRVnextchance
                                        |-BSRVwrapup
                                            |-BSRVnomore
```

FIGURE 2.5-1. Top-Level Sensors Code Tree.

```
BSRVevent
    |-TMemory::Index2Ptr
    |-BSRVinitialize
    |   |-TMemory::Index2Ptr
    |   |-TMemory::Allocate
    |   |   \-TMemory::DoAllocate
    |   |       |-TMemory::GetBlockLength
    |   |       |-CountMemOpns
    |   |       |   \-TMaster::DebugOn
    |   |       |-TMemory::Ptr2Index
    |   |       |-ProgramStop::ProgramStop
    |   |       |   \-SwegExcpt::SwegExcpt
    |   |       |       \-StringDup
    |   |       \-TMemory::WriteSummary
    |   |           |-TMemory::WordsUsed
    |   |           |-TMemory::CalcWdsLeft
    |   |           \-CountMemOpns
    |   |-TMemory::Ptr2Index
    |   |-TMemory::LLSTsearch
    |   |   |-TMemory::LLSTsearch
    |   |   |   \-TMemory::Index2Ptr
    |   |   \-TMemory::Index2Ptr
    |   |-snropmode
    |   |   |-TMemory::Ptr2Index
    |   |   \-TMemory::LLSTsearchhard
    |   |       |-TMemory::LLSTsearchhard
    |   |       |   \-TMemory::Index2Ptr
    |   |       \-TMemory::Index2Ptr
    |   |-snropfreq
    |   \-TMaster::GetPlatform
    |       \-TAccDirect::GetItem
    |           \-TAccDirect::GetItem
    |-TMemory::Ptr2Index
    |-redwood
    |   |-TMemory::Index2Ptr
    |   \-TMemory::Ptr2Index
    |-TMemory::LLSTremove
    |   |-TMemory::Index2Ptr
    |   |-TMemory::LLSTsearch
    |   |   \-TMemory::Index2Ptr
    |   \-TMemory::Deallocate
    |       |-TMemory::Ptr2Index
    |       |-TMemory::DeallocFront
    |       |   \-TMemory::GetBlockLength
    |       |-CountMemOpns
    |       \-TMemory::RcylBlock
    |           |-TMemory::Index2Ptr
    |           \-TMemory::Ptr2Index
```

FIGURE 2.5-2. Sensors Code Tree.

```
|-TMemory::LLSTsearch
|-TMemory::LLSTsearchhard
|-BSRVonechance
|    |-TMemory::Index2Ptr
|    |-WhereIsIt::CalcPosition
|    |-TMemory::LLSTlength2
|    |    |-TMemory::LLSTlength2
|    |    |    \-TMemory::Index2Ptr
|    |    \-TMemory::Index2Ptr
|    |-TMemory::Allocate
|    |-BSRVsignature
|    |    |-TMemory::Index2Ptr
|    |    |-operator-
|    |    |-WhereIsIt::CalcPosition
|    |    |-DVector::Norm
|    |    |-BSRVaimpt
|    |    |    |-DVector::DVector
|    |    |    |-dist
|    |    |    |-DVector::Getx
|    |    |    |-DVector::Gety
|    |    |    |-DVector::Getz
|    |    |    |-dbg_atan2
|    |    |    |    \-MathExcpt::MathExcpt
|    |    |    |         \-SwegExcpt::SwegExcpt
|    |    |    |-WhereIsIt::CalcPosition
|    |    |    |-DVector::Putz
|    |    |    |-DVector::Norm
|    |    |    |-FloatVector::operator=
|    |    |    |    |-DVector::Getx
|    |    |    |    |-DVector::Gety
|    |    |    |    \-DVector::Getz
|    |    |    |-operator-
|    |    |    |    |-DVector::DVector
|    |    |    |    |-DVector::Getx
|    |    |    |    |-DVector::Gety
|    |    |    |    \-DVector::Getz
|    |    |    |-DVector::GetHorizLength
|    |    |    \-FloatVector::Putz
|    |    |-DVector::operator+=
|    |    |-TMemory::Ptr2Index
|    |    |-WhereIsIt::CalcUnitVel
|    |    |-WhereIsIt::CalcUpVector
|    |    |-DVector::VecAng
|    |    |    |-operator^
|    |    |    |-operator*
|    |    |    |-DVector::DVector
|    |    |    |-DVector::GetLength
|    |    |    |-isZeroEquiv
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|   |   |   \-dbg_atan2
|   |   |-TTable::SearchInt
|   |   |   \-TMemory::Ptr2Index
|   |   |-BSRVgetsigtable
|   |   |   |-TMemory::Index2Ptr
|   |   |   \-srhpro
|   |   |-ergazel
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-srhpro
|   |   |   \-TMemory::Ptr2Index
|   |   |-BSRVfiresig
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-TTable::SearchInt
|   |   |   |-srhpro
|   |   |   |-TMemory::Ptr2Index
|   |   |   |-dbg_pow
|   |   |   |   \-MathExcpt::MathExcpt
|   |   |   |-ergazel
|   |   |   \-TMemory::LLSTremove
|   |   |-WhereIsIt::CalcSpeed
|   |   |-sigmov
|   |   |   |-TMemory::Index2Ptr
|   |   |   \-srhpro
|   |   \-FloatVector::operator=
|   |-TMemory::Deallocate
|   |   |-TMemory::DeallocFront
|   |   |-TMemory::Index2Ptr
|   |   |-CountMemOpns
|   |   \-TMemory::RcylBlock
|   |-operator-
|   |-DVector::GetLength
|   |-WhereIsIt::CalcVelocity
|   |-operator^
|   |-BSRVmasking
|   |   |-TMemory::Index2Ptr
|   |   |-WhereIsIt::CalcPosition
|   |   |-operator-
|   |   |-DVector::GetHorizLength
|   |   |-srhpro
|   |   |-DVector::Getz
|   |   |-DVector::DVector
|   |   |   |-FloatVector::Getx
|   |   |   |-FloatVector::Gety
|   |   |   \-FloatVector::Getz
|   |   |-WhereIsIt::CalcFwdVector
|   |   |-WhereIsIt::CalcUpVector
|   |   |-DVector::CrossProduct
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|   |   |-DVector::Rotated
|   |   |   |-DVector::DVector
|   |   |   |-DVector::Getx
|   |   |   |-DVector::Gety
|   |   |   \-DVector::Getz
|   |   |-DVector::VecAng
|   |   |-loschk
|   |   |   |-TMaster::GetTerrain
|   |   |   |-TMaster::TerrainOn
|   |   |   |-DVector::Getz
|   |   |   |-dbg_acos
|   |   |   |   |-MathExcpt::MathExcpt
|   |   |   |   \-acos_c
|   |   |   |-operator-
|   |   |   |-DVector::GetHorizLength
|   |   |   |-DVector::Getx
|   |   |   |-DVector::Gety
|   |   |   \-TTerrain::LineOfSight
|   |   |       |-DVector::Getz
|   |   |       |-DVector::Getx
|   |   |       |-DVector::Gety
|   |   |       |-dist
|   |   |       |-dbg_acos
|   |   |       |-VertexIndex::VertexIndex
|   |   |       |   \-VertexIndex::operator=
|   |   |       |-TTerrain::FindTriangle
|   |   |       |   |-VertexIndex::VertexIndex
|   |   |       |   |-TAddress::Cartesian2Spherical
|   |   |       |   |   |-dbg_sqrt
|   |   |       |   |   |-dbg_asin
|   |   |       |   |   |   |-MathExcpt::MathExcpt
|   |   |       |   |   |   \-asin_c
|   |   |       |   |   \-dbg_atan2
|   |   |       |   |-TMessages::WriteMessage
|   |   |       |   |   |-TMessages::GetMsg
|   |   |       |   |   |-TMessages::PrintALine
|   |   |       |   |   |   \-TSeqFile::Write
|   |   |       |   |   |       \-MFiles::Append
|   |   |       |   |   \-sysdun
|   |   |       |   |       |-TActWindow::GetNext
|   |   |       |   |       |-TActWindow::~TActWindow
|   |   |       |   |       \-ProgramStop::ProgramStop
|   |   |       |   |-TAddress::Spherical2Cartesian
|   |   |       |   |   \-TMaster::DebugOn
|   |   |       |   |-VertexIndex::operator=
|   |   |       |   |-VerticeArray::operator[]
|   |   |       |   |   \-VertexIndex::Value
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|   |   |       |   |-TTerrain::toPtr
|   |   |       |   |   \-SwegExcpt::SwegExcpt
|   |   |       |   |-operator-
|   |   |       |   |   \-VertexIndex::VertexIndex
|   |   |       |   |-VertexIndex::operator++
|   |   |       |   |-dist
|   |   |       |   |-operator+
|   |   |       |   |   \-VertexIndex::VertexIndex
|   |   |       |   \-VertexIndex::operator+=
|   |   |       |       \-operator+
|   |   |       \-TTerrain::EdgeMasklos
|   |   |           |-dist
|   |   |           |-VerticeArray::operator[]
|   |   |           |-operator+
|   |   |           |-VertexIndex::operator+=
|   |   |           |-isZeroEquiv
|   |   |           \-TTerrain::toIndex
|   |   |               \-SwegExcpt::SwegExcpt
|   |   \-featlos
|   |       |-TMaster::GetTerrain
|   |       |-TMaster::TerrainOn
|   |       |-DVector::Getx
|   |       |-DVector::Gety
|   |       |-DVector::Getz
|   |       |-TTerrain::Elevation
|   |       |   |-DVector::Getx
|   |       |   |-DVector::Gety
|   |       |   |-VertexIndex::VertexIndex
|   |       |   |-TTerrain::FindTriangle
|   |       |   |-VerticeArray::operator[]
|   |       |   |-operator+
|   |       |   \-isZeroEquiv
|   |       |-TMemory::Index2Ptr
|   |       |-TMemory::AllocTemp
|   |       |   \-TMemory::DoAllocate
|   |       |-sorted_collection::sorted_collection
|   |       |-TAddress::GetAddresses
|   |       |   |-TAddress::GetCode
|   |       |   |   |-DVector::Getx
|   |       |   |   |-DVector::Gety
|   |       |   |   \-TMessages::WriteMessage
|   |       |   |-TAddress::InsertVertCodes
|   |       |   |   |-sorted_collection::insert_nodup
|   |       |   |   |   |-MTree::insert_nodup
|   |       |   |   |   |   |-MTree::insert_nodup
|   |       |   |   |   |   \-MTree::MTree
|   |       |   |   |   \-MTree::MTree
|   |       |   |   \-numerical
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|   |       |   |-operator-
|   |       |   |-operator^
|   |       |   |-TAddress::GetCellRadius
|   |       |   |-dbg_sqrt
|   |       |   |-operator*
|   |       |   |-DVector::operator+=
|   |       |   \-operator+
|   |       |-sorted_collection::getfirst
|   |       |-sorted_collection::getnext
|   |       |   \-MTree::getnext
|   |       |-TAddress::GetShapeList
|   |       |   |-TAddrNode::DataPresent
|   |       |   |   \-TAddrNode::GetNode
|   |       |   |       |-TAddrData::GetCode
|   |       |   |       |-TAddrData::TAddrData
|   |       |   |       |-TAddrNode::TAddrNode
|   |       |   |       |   \-MTree::MTree
|   |       |   |       \-TAddrData::PutNodePtr
|   |       |   \-TAddrData::GetShapeList
|   |       |       \-TMemory::Index2Ptr
|   |       |-TMemory::Ptr2Index
|   |       |-TTable::SearchInt
|   |       |-region
|   |       |   |-DVector::Getx
|   |       |   |-DVector::Gety
|   |       |   |-dbg_atan2
|   |       |   \-dist
|   |       |-crslwp
|   |       |   |-DVector::DVector
|   |       |   |-DVector::DVector
|   |       |   |-dbg_sqrt
|   |       |   |-crslwc
|   |       |   |   |-operator-
|   |       |   |   |-DVector::Putz
|   |       |   |   |-operator^
|   |       |   |   |-dbg_sqrt
|   |       |   |   |-operator+
|   |       |   |   \-operator*
|   |       |   |-TMemory::Index2Ptr
|   |       |   |-DVector::operator
|   |       |   |-crslwl
|   |       |   |   |-operator-
|   |       |   |   |-operator*
|   |       |   |   |-DVector::Getz
|   |       |   |   |-operator+
|   |       |   |   \-operator*
|   |       |   |-TMemory::Allocate
|   |       |   |-DVector::Getx
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|   |       |   |-DVector::Gety
|   |       |   \-TMemory::Ptr2Index
|   |       |-DVector::DVector
|   |       |-sorted_collection::delete_collection
|   |       |   \-MTree::delete_tree
|   |       |       |-MTree::delete_tree
|   |       |       \-TMemory::Deallocate
|   |       |           |-TMemory::Deallocate
|   |       |           \-TMemory::Ptr2Index
|   |       \-TMemory::Deallocate
|   |-WhereIsIt::CalcSpeed
|   |-srhpro
|   |-DVector::Getz
|   |-DVector::GetHorizLength
|   |-ergatn
|   |   \-srhpro
|   |-BSRVcalculate
|   |   |-jamcal
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-WhereIsIt::CalcPosition
|   |   |   |-TAddrData::GetParentData
|   |   |   |   \-TAddrNode::GetParentData
|   |   |   |-TAddrData::GetJamInteractions
|   |   |   |-TTable::SearchInt
|   |   |   \-TMemory::Deallocate
|   |   |-operator-
|   |   |-WhereIsIt::CalcPosition
|   |   |-operator^
|   |   |-dbg_pow
|   |   |-BSRVtgtsignal
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-erggar
|   |   |   |   |-antgeom
|   |   |   |   |   |-DVector::DVector
|   |   |   |   |   |-WhereIsIt::CalcPosition
|   |   |   |   |   |-DVector::operator=
|   |   |   |   |   |   |-FloatVector::Getx
|   |   |   |   |   |   |-FloatVector::Gety
|   |   |   |   |   |   \-FloatVector::Getz
|   |   |   |   |   |-DVector::DVector
|   |   |   |   |   |-operator-
|   |   |   |   |   |-DVector::Norm
|   |   |   |   |   |-DVector::GetHorizLength
|   |   |   |   |   |-DVector::operator
|   |   |   |   |   |-DVector::Getx
|   |   |   |   |   |-DVector::Getz
|   |   |   |   |   |-DVector::Gety
|   |   |   |   |   |-WhereIsIt::CalcUnitVel
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|   |   |   |   |   |-WhereIsIt::CalcUpVector
|   |   |   |   |   |-operator*
|   |   |   |   |   |-operator+
|   |   |   |   |   |-operator*
|   |   |   |   |   |-FloatVector::Getx
|   |   |   |   |   |-FloatVector::Gety
|   |   |   |   |   |-FloatVector::Getz
|   |   |   |   |   |-DVector::VecAng
|   |   |   |   |   \-DVector::GetLength
|   |   |   |   |-TMemory::Index2Ptr
|   |   |   |   |-srhpro
|   |   |   |   |-TMemory::Ptr2Index
|   |   |   |   \-ergazel
|   |   |   |-operator-
|   |   |   |-WhereIsIt::CalcVelocity
|   |   |   |-DVector::GetLength
|   |   |   |-WhereIsIt::CalcPosition
|   |   |   |-operator^
|   |   |   |-operator/
|   |   |   |-srhpro
|   |   |   \-dbg_pow
|   |   \-BSRVsystsensing
|   |       |-TMemory::Index2Ptr
|   |       |-erggar
|   |       |-WhereIsIt::CalcPosition
|   |       |-DVector::Getz
|   |       |-operator-
|   |       |-DVector::GetHorizLength
|   |       |-TMemory::Ptr2Index
|   |       |-TMemory::LLSTsearch
|   |       |-TTable::SearchInt
|   |       |-jamcal
|   |       |-ergatn
|   |       |-TMemory::Allocate
|   |       \-TMemory::Deallocate
|   |-TMemory::Ptr2Index
|   |-TMemory::LLSTsearch
|   |-to_ptr
|   |-DVector::VecAng
|   |   |-DVector::DVector
|   |   \-DVector::VecAng
|   |-BSRVresults
|   |   |-TMaster::GetTerrain
|   |   |-TMemory::Index2Ptr
|   |   |-TMemory::Deallocate
|   |   |-TMemory::Allocate
|   |   |-WhereIsIt::CalcPosition
|   |   |-DVector::Getx
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|   |   |-DVector::Gety
|   |   |-DVector::Getz
|   |   |-TMaster::TerrainOn
|   |   |-TTerrain::Elevation
|   |   |-BSRVsnrerrors
|   |   |   |-isZeroEquiv
|   |   |   |-RNDMgrn
|   |   |   |   |-RNDMurn
|   |   |   |   |   \-TMaster::GetRandomTable
|   |   |   |   |       \-TMemory::Index2Ptr
|   |   |   |   |-dist
|   |   |   |   \-dbg_sqrt
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-antgeom
|   |   |   |-WhereIsIt::CalcUnitVel
|   |   |   |-WhereIsIt::CalcUpVector
|   |   |   |-operator*
|   |   |   |-operator+
|   |   |   |-operator*
|   |   |   |-DVector::Norm
|   |   |   |-WhereIsIt::CalcPosition
|   |   |   |-DVector::Getz
|   |   |   |-DVector::Getx
|   |   |   \-DVector::Gety
|   |   |-WhereIsIt::CalcSpeed
|   |   |-WhereIsIt::CalcHeading
|   |   |-WhereIsIt::CalcPitch
|   |   |-TMemory::Ptr2Index
|   |   \-TMemory::LLSTsearch
|   |-TTable::SearchInt
|   |-RNDMurn
|   \-BSRVtrack
|       |-TMemory::Index2Ptr
|       |-RNDMurn
|       |-TMemory::Ptr2Index
|       |-TMaster::GetPlatform
|       |-immol8
|       |   |-injure
|       |   |   |-TMemory::Index2Ptr
|       |   |   |-injsnr
|       |   |   |   |-TMemory::Index2Ptr
|       |   |   |   |-TMemory::Ptr2Index
|       |   |   |   |-MITRcontrol
|       |   |   |   |-jetind
|       |   |   |   |   |-TMemory::LLSTremove
|       |   |   |   |   \-TMemory::Deallocate
|       |   |   |   |-TMemory::Deallocate
|       |   |   |   \-TMemory::LLSTremove
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|       |   |   |-injcom
|       |   |   |   |-TMemory::Index2Ptr
|       |   |   |   |-MITRcontrol
|       |   |   |   |-TMemory::Deallocate
|       |   |   |   |-jetind
|       |   |   |   \-TMemory::LLSTremove
|       |   |   |-delswt
|       |   |   |-MITRcontrol
|       |   |   |-jetind
|       |   |   |-TMemory::Deallocate
|       |   |   |-TMemory::LLSTremove
|       |   |   \-badpro
|       |   |       |-TMemory::Index2Ptr
|       |   |       |-redwood
|       |   |       |-TMemory::LLSTremove
|       |   |       |-limpendq
|       |   |       |   \-TMemory::Deallocate
|       |   |       |-TMemory::Deallocate
|       |   |       |-TMemory::Ptr2Index
|       |   |       \-limevent
|       |   |-mutil8
|       |   |   |-TMemory::Index2Ptr
|       |   |   |-haltit
|       |   |   |   |-TMemory::Index2Ptr
|       |   |   |   |-TMemory::Ptr2Index
|       |   |   |   |-TMemory::Deallocate
|       |   |   |   |-TPathEntry::GetBeforePoint
|       |   |   |   \-TPathEntry::SetTDepart
|       |   |   |-TMemory::LLSTremove
|       |   |   |-TAddrData::DelOccupant
|       |   |   |   |-TMemory::LLSTremove
|       |   |   |   \-TAddrData::TryDeletion
|       |   |   |-TMemory::LLSTsearch
|       |   |   |-TMemory::Ptr2Index
|       |   |   |-limevent
|       |   |   |-yaeail
|       |   |   |-TMemory::Deallocate
|       |   |   |-TMemory::LLSTlistofflist
|       |   |   \-TMaster::GetPlatform
|       |   |-TMemory::Ptr2Index
|       |   |-ailrip
|       |   |   |-TMemory::Index2Ptr
|       |   |   |-TMaster::GetPlayer
|       |   |   |-TPlayer::IsAlive
|       |   |   |-TMemory::Allocate
|       |   |   |-TMemory::Ptr2Index
|       |   |   \-yaeail
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|         |    |-TMemory::Index2Ptr
|         |    \-sasin8
|         |         |-TMemory::Index2Ptr
|         |         |-TMaster::GetEnvironment
|         |         |    \-TMemory::Index2Ptr
|         |         |-TMemory::Ptr2Index
|         |         |-diefri
|         |         |    |-TMemory::Ptr2Index
|         |         |    |-TMemory::LLSTsearch
|         |         |    |-TMemory::Deallocate
|         |         |    |-TMemory::LLSTremove
|         |         |    |-jetind
|         |         |    \-TMemory::Index2Ptr
|         |         |-KILLperception
|         |         |    |-TMemory::Ptr2Index
|         |         |    |-TMemory::Index2Ptr
|         |         |    \-begone
|         |         |         |-TMemory::LLSTremove
|         |         |         |-TMemory::Deallocate
|         |         |         |-delswt
|         |         |         |-MITRcontrol
|         |         |         \-TMemory::Index2Ptr
|         |         |-TMemory::Deallocate
|         |         |-TMemory::Deallocate
|         |         |-limevent
|         |         |-TMemory::LLSTremove
|         |         |-redwood
|         |         \-limpendq
|         |-TMemory::LLSTremove
|         \-yaeail
|-BSRVnextchance
|    |-TMemory::LLSTsearchhard
|    |-TMemory::Index2Ptr
|    |-operator-
|    |-WhereIsIt::CalcPosition
|    |-DVector::GetHorizLength
|    |-TMemory::Ptr2Index
|    |-nayist
|    \-redwood
|-TMemory::Allocate
|-yaeail
|-TMemory::Deallocate
\-BSRVwrapup
         |-TMemory::Index2Ptr
         |-TMemory::Deallocate
         |-TMBRpercplat
         |    \-TMemory::Deallocate
         |-TMemory::Allocate
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

```
|-yaeail
|-TMemory::LLSTsearch
|-TMemory::Deallocate
|-TMemory::Ptr2Index
\-BSRVnomore
    |-TMemory::Index2Ptr
    |-TMemory::Ptr2Index
    |-TMaster::GetPlatform
    |-TMemory::Deallocate
    |-TMemory::LLSTsearch
    \-TMemory::Deallocate
```

FIGURE 2.5-2. Sensors Code Tree. (Contd.)

2.5.4 Assumptions and Limitations

- Energy transmission is instantaneous.
- The speed of light is 299,792,800 m/sec and the speed of sound is 330.28 m/sec.
- Energy use is not explicitly represented.
- The level of detail in sensor-derived perceptions is the same as the level of detail that was defined by the user in the perceived platform.
- Locations of detected targets are accurately perceived.
- Signal polarization does not affect detection.
- Target fluctuation and signal integration do not affect detection.
- Root mean square signal power is used in all calculations.
- Received power is uniformly distributed over the receiver bandwidth.
- IFF is performed at each detection if a PLAYER has IFF capability.
- IFF classification of a target depends on only the SIDEs of the sensor and the target.
- Explicit S/I calculations must be performed to represent sensor operations.
- SWEG allows the user to select the data items to be collected by a sensor type, although currently, all sensors are implicitly assumed to collect altitude and planar position.
- All sensor perceptions are assumed to be perfect at the time of sensing.
- User-specified maximum and minimum slew angles for sensor elevation are allowed; however, slew angles in azimuth and slew rates in both azimuth and elevation are not.

2.5.5 Known Problems or Anomalies

None.